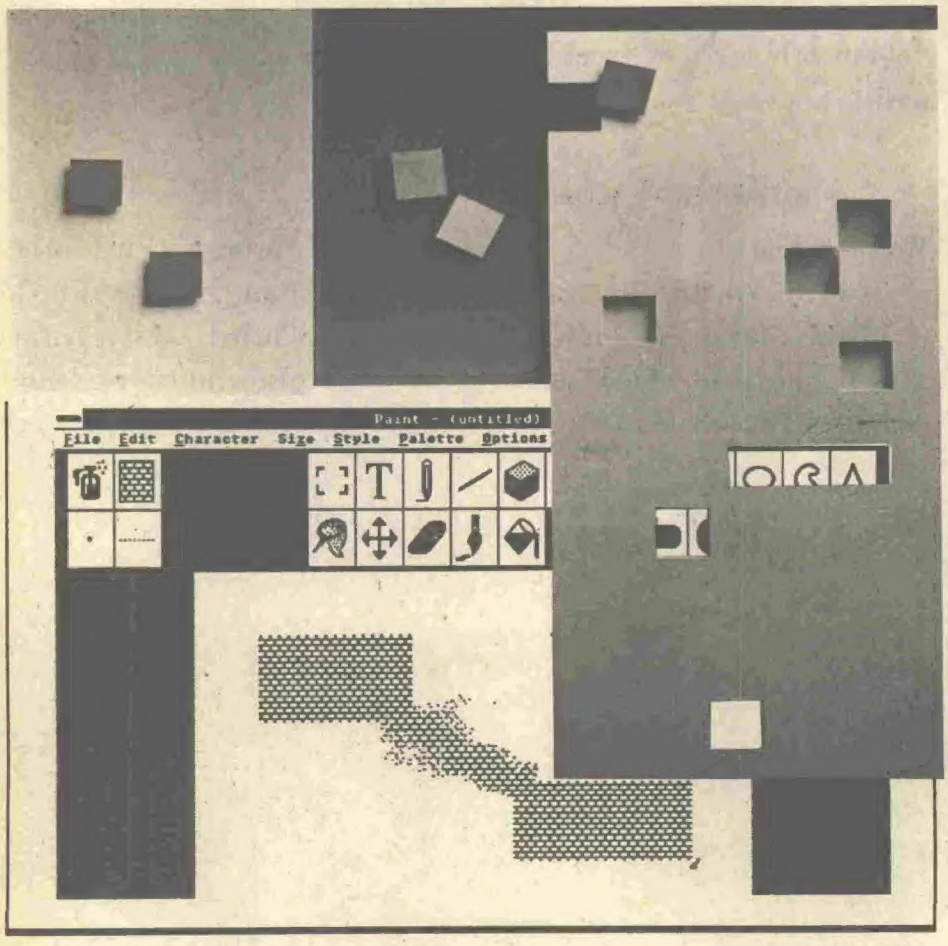
SOFT MAIL

เขาพัฒนาโปรแกรมภายใต้ไมโครซอฟต์วินโดว์กันอย่างไร

เรียนรู้ลักษณะโครงสร้างระบบวินโดว์และการพัฒนาโปรแกรมภายใต้วินโดว์ของไมโครซอฟต์



โปรแกรมวินโดว์ แตกต่างจากโปร
แกรมส่วนมากที่ถูกพัฒนาเพื่อทำงา
ภายใต้เอ็มเอสดอส เช่น มีเมนูและเมา
แทนการพิมพ์คำสั่ง อาจจะมีการเปลี่ยน
แปลงพื้นที่แสดงผลตามแต่ผู้ใช้จะเลือ
โปรแกรมมาทำงาน โปรแกรมวินโดว์จ
เรียกฟังก์ชันของระบบวินโดว์ที่กำหน
ให้เรียกใช้ มีการถ่ายเทข้อมูลระหว่า
โปรแกรม นอกจากนั้น ระบบวินโด
ทำให้โครงสร้างของโปรแกรมเปลี่ยนไ
เพื่อที่จะทำให้หลาย ๆ โปรแกรมสามาร
ที่จะใช้โปรเซสเซอร์ร่วมกันและทำงา
พร้อม ๆ กัน

เมื่อมองจากมุมของผู้ใช้ แต่ล
โปรแกรมจะทำงานอยู่ในกรอบสี่เหลี่ย
ที่เรียกว่า "วินโดว์" เมื่อโปรแกรมเริ่
ทำงานวินโดว์ก็จะเกิดขึ้น เมื่อผู้ใช้ป
วินโดว์โปรแกรมก็จะหยุดทำงาน เมื่อ
ใช้คลิกที่วินโดว์โปรแกรมก็จะกระท
งานบางอย่าง จากมุมมองของนักเขีย
โปรแกรมก็เช่นเดียวกัน แต่ละโปรแกร
ต้องการกระบวนความ (procedur
เฉพาะ ที่ทำงานเกี่ยวกับวินโดว์ โป
แกรมไม่ควรจะแสดงผลออกนอกกรอ
วินโดว์ ข้อมูลจากเมาส์และคีย์บอร์
จะถูกส่งไปยังวินโดว์ที่กำหนด ในท
กลับกันวินโดว์ของโปรแกรมนั้น ๆ
มีรูปแบบที่แน่นอน และการเขียนโป
แกรมวินโดว์หมายถึงการกำหนดพฤ
กรรมของวินโดว์นั้น ๆ เรามองเห็
อะไรในวินโดว์บ้าง

ส่วนประกอบภายนอกบนวินโดว์

Border เป็นเส้นกรอบของวินโดว์ การคลิกบนบอร์เดอร์และลากเมาส์จะเป็นการย้ายหรือเปลี่ยนขนาดของวินโดว์ ดังรูปที่ 1

Title Bar เป็นส่วนที่แสดงชื่อของโปรแกรม ปกติโปรแกรมที่มีการกระทำเกี่ยวกับไฟล์จะมีการแสดงชื่อไฟล์บนไตเติลบาร์นี้ด้วย

Control Box เป็นสี่เหลี่ยมเล็ก ๆ ที่อยู่ตรงมุมบนซ้ายของวินโดว์ การคลิกที่คอนโทรลบ็อกจะเป็นการเปิดเมนูควบคุม (control menu)

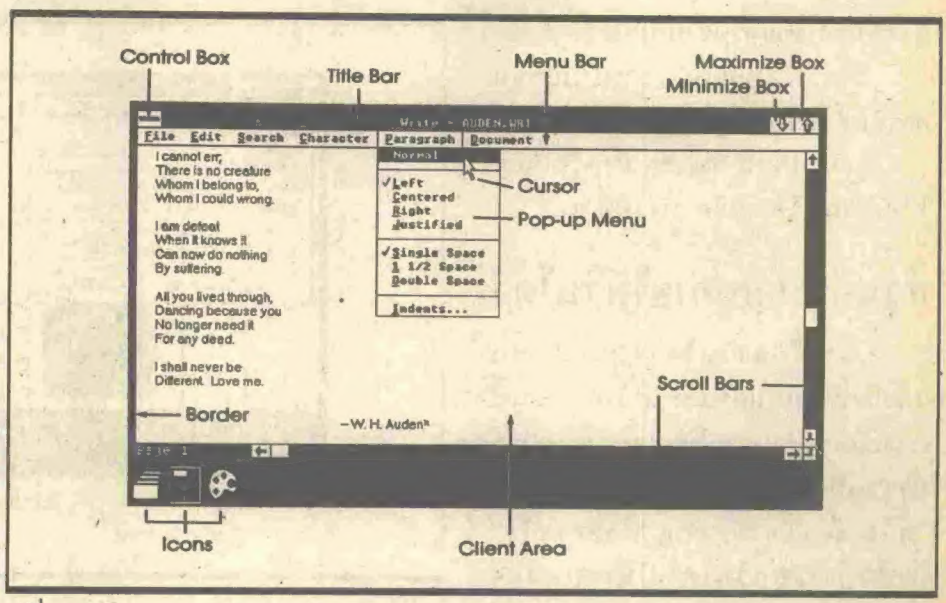
Maximize และ Minimize Box เป็นสี่เหลี่ยมที่อยู่ทางด้านมุมบนขวาของวินโดว์ และเป็นสี่เหลี่ยมที่มีลูกศรชี้ขึ้นและลงภายในสี่เหลี่ยม การคลิกที่สี่เหลี่ยมที่มีลูกศรขึ้นทำให้วินโดว์ขยายจนเต็มจอ และการคลิกสี่เหลี่ยมที่มีลูกศรชี้ลงเป็นการทำให้วินโดว์ลดขนาดเป็นสัญลักษณ์รูปภาพเล็ก ๆ ที่เรียกว่า ไอคอน (icon)

Menu Bar เป็นส่วนที่แสดงข้อเลือกหรือคำสั่งต่าง ๆ ที่โปรแกรมกำหนดให้ผู้ใช้เรียกใช้ได้โดยการเลื่อนเมาส์ไปที่ข้อเลือกนั้น ๆ และคลิกเมาส์

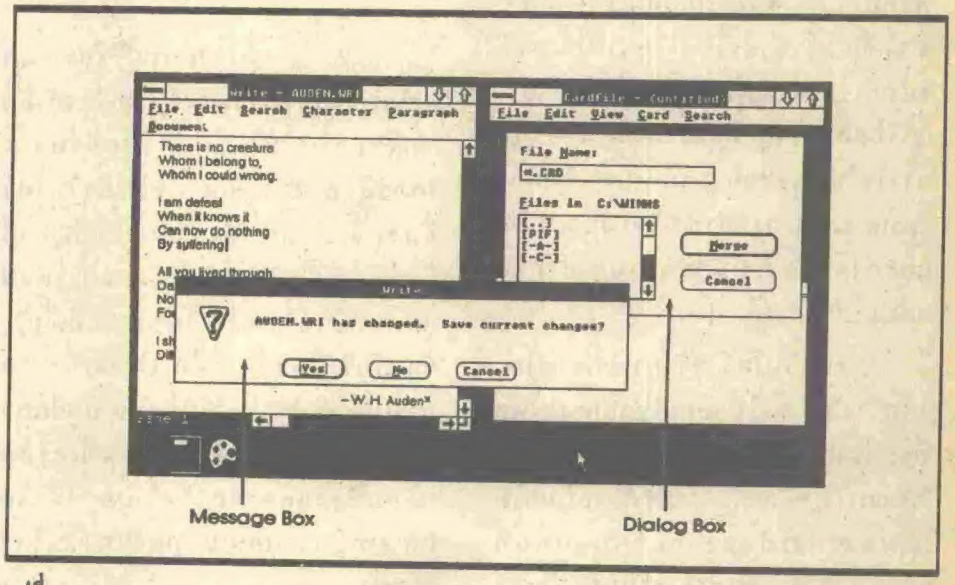
Pop Up Menu เป็นเมนูของข้อเลือกย่อย จะใช้เมื่อมีคำสั่งหรือข้อเลือกที่เกี่ยวข้องกัน

Scroll Bar เป็นส่วนที่ให้ผู้ใช้เลื่อนพื้นที่แสดงผลใน scroll bar มีกล่องสี่เหลี่ยมเล็ก ๆ เรียกว่า "scroll box" การคลิกบนสี่เหลี่ยมนี้และลากขึ้นและลง ซ้ายหรือขวา จะเป็นการเลื่อนพื้นที่แสดงผล

Client Area เป็นส่วนที่เหลือของวินโดว์ที่ให้เป็นส่วนแสดงผลของโปรแกรมวินโดว์



รูปที่ 1



รูปที่ 2

Message Box เป็นส่วนที่แจ้งผู้ใช้เกี่ยวกับข่าวสารหรือมีการทำงานบางอย่างผิดพลาด อันที่จริงแล้วเมสเสจบ็อกไม่ได้เป็นส่วนหนึ่งของวินโดว์ เพราะว่าตัวมันเองเป็นวินโดว์อยู่แล้ว เช่นมีบอร์เดอร์และไตเติลบาร์ แต่ว่าไม่มีเมนูและปุ่ม (button) ที่จัดการเกี่ยวกับวินโดว์ทั่วไป ดังรูปที่ 2

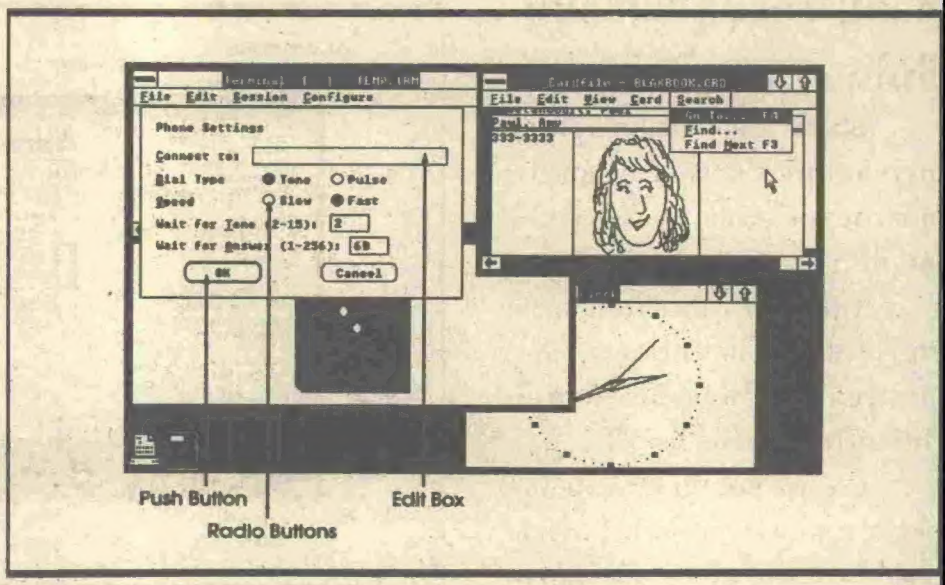
Dialog Box เป็นวินโดว์อีกชนิดหนึ่งที่โปรแกรมวินโดว์สร้างขึ้น ไดอะล็อกบ็อกถูกใช้ขึ้นเพื่อขอข่าวสารหรือข้อมูลเพิ่มเติมจากผู้ใช้ เช่น จะเปิดไฟล์ไหน จะวาดรูปด้วยสีอะไร โดยส่วนใหญ่แล้วไดอะล็อกบ็อก ยังประกอบไปด้วยส่วนเล็ก ๆ อีกหลายส่วน คือ

Edit Box เป็นส่วนที่รับข้อความ

การพิมพ์หรือแก้ไขจากคีย์บอร์ด

Radio Button สำหรับเปลี่ยนการติดตั้งหรือข้อมูลของโปรแกรม

และ Push Button สำหรับเลือกให้ทำงานหรือยกเลิก ดังรูปที่ 3



รูปที่ 3

ส่วนประกอบภายในวินโดว์

เราได้มองวินโดว์จากภายนอกแล้วว่าประกอบด้วยอะไรบ้าง ตอนนี้เราจะมองดูส่วนภายในของระบบวินโดว์บ้างว่าเป็นอย่างไร เมื่อมองเข้าไปภายในก็จะต้องเกี่ยวข้องกับ "มัลติทาสกิ้ง" เพราะว่าระบบวินโดว์เป็นระบบหนึ่งที่ยอมให้หลาย ๆ โปรแกรมทำงานไปพร้อม ๆ กัน ซึ่งทำให้เกิดปัญหาหลัก ๆ 3 ข้อคือ การจัดสรรแบ่งจอภาพให้แต่ละโปรแกรมเพื่อที่ผลลัพธ์ของแต่ละโปรแกรมแสดงได้ถูกต้องและไม่ยุ่งเกี่ยวกัน การใช้โปรเซสเซอร์ร่วมกันเพื่อที่แต่ละโปรแกรมสามารถทำงานไปพร้อม ๆ กัน และการจัดสรรหน่วยความจำสำหรับแต่ละโปรแกรม

ระบบวินโดว์ได้จัดการจอภาพโดยได้สร้างวินโดว์ให้แต่ละโปรแกรม เพื่อที่จะป้องกันไม่ให้การแสดงผลของแต่ละโปรแกรมยุ่งเกี่ยวกัน และดังนั้นโปรแกรมก็ไม่ควรที่จะแสดงผลออกนอกพื้นที่ของวินโดว์ของตัวเอง พร้อมกันนั้นตัวระบบวินโดว์เองจะตัดหรือ "คลิบ (clip)" เมื่อมีคำสั่งที่แสดงผลเกินบริเวณขอบเขตของวินโดว์นั้น ๆ และจะยอมให้แสดงผลเฉพาะส่วนของไคลต์แอเรีย (Client Area) ที่สามารถมองเห็นเท่านั้น (คือไม่ถูกบังด้วยวินโดว์อื่น) แม้กระนั้นการคลิบก็ยังไม่เพียงพอต่อการจัดการจอภาพ เพราะว่าการคลิบเป็นเพียงแค่การป้องกันการแสดงนอกบริเวณของวินโดว์ แต่ไม่ได้ป้องกันการถูกทับจากวินโดว์อื่น ๆ เช่น การที่ผู้ใช้สามารถเปลี่ยนขนาดของวินโดว์หรือเรียงลำดับ (stack) ของวินโดว์ใหม่ หรือส่วนของไดอะล็อกและเมสเสจบ็อกของวินโดว์อื่น ๆ ขึ้นมาทับ ดังนั้นการแก้ปัญหาก็คือว่า เมื่อไรก็ตามที่มีบริเวณบางส่วนถูกทับหรือทำลายไป โปรแกรมที่เกี่ยวข้องกับวินโดว์นั้นจะต้องแสดงหรือวาดส่วนนั้นเสียใหม่ ดังนั้นโดยปกติแล้วโปรแกรมวินโดว์ต้องมีส่วนที่เตรียมพร้อมที่จะแสดงจอภาพใหม่ เมื่อใดก็ตามที่มีการเปลี่ยนแปลงของวินโดว์เกิดขึ้น

เมสเสจ (Message)

ระบบวินโดว์จัดการให้โปรแกรมทำงานแบบขนาน โดยใช้การส่งเมสเสจ (Message) ไปยังโปรแกรมแต่ละโปรแกรม เมสเสจเป็นโครงสร้างข้อมูลอันหนึ่งที่บรรจุข่าวสารเกี่ยวกับการกระทำหรือเหตุการณ์บางอย่างที่เกิดขึ้น หรือต้องการที่จะให้เกิดขึ้น เช่น เมื่อผู้ใช้คลิกเมาส์ที่วินโดว์ ระบบวินโดว์จะส่งเมสเสจที่เรียกว่า "button down" ไปยังโปรแกรมที่เป็นเจ้าของวินโดว์นั้น ในแต่ละเมสเสจจะประกอบด้วยส่วนต่าง ๆ อย่างน้อยที่สุด 2 ส่วนคือ หมายเลขของวินโดว์ที่เกี่ยวข้อง และหมายเลขที่กำหนดเหตุการณ์ เพื่อให้ง่ายต่อการอ่านโปรแกรม และสำหรับโปรแกรมเมอร์ได้มีการกำหนดชื่อของเมสเสจแทนหมายเลข เช่น WM_PAINT (WM มาจาก Window Message) แทนหมายเลข 0x000F ชื่อทั้งหมดของเมสเสจและหมายเลขจะอยู่ในไฟล์ชื่อ window.h ถ้าสำรวจดูเมสเสจในไฟล์นี้จะเห็นว่ามีมากกว่า 100 เมสเสจ ซึ่งโดยปกติแล้วโปรแกรมจะรับบางเมสเสจไปทำงานเท่านั้น นอกนั้นจะผ่านกลับไปให้ระบบวินโดว์จัดการแทน

เมสเสจมาได้จากหลายแหล่ง บางส่วนมาจากผู้ใช้ นั่นคือเมสเสจที่เกี่ยวข้องกับคีย์บอร์ด เมาส์ scroll bar และเมนู ตัวระบบวินโดว์เองก็สร้างเมสเสจขึ้นเช่นกัน เช่นเมื่อบริเวณในไคลต์แอเรีย

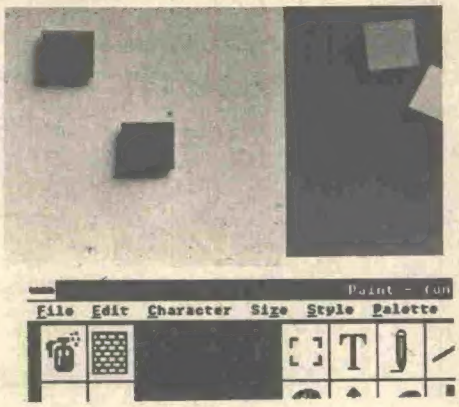
ถูกทำลายหรือรบกวนระบบวินโดว์จะส่งเมสเสจ WM_PAINT ไปยังโปรแกรมวินโดว์นั้น เพื่อสร้างส่วนของไคลต์แอเรียสมัยใหม่ เมสเสจที่เข้ามาในระบบจะถูกจัดอยู่ในรูปเรียงลำดับก่อนหลังตามคิวภายใน queue ของระบบ ก่อนที่จะถูกส่งไปยัง queue ของแต่โปรแกรม

การอินพุตข้อมูล

โปรแกรมวินโดว์จะไม่รับอินพุตของอุปกรณ์โดยตรงอย่างโปรแกรมทั่วไป ระบบวินโดว์จะเป็นผู้ส่งอินพุต (เมสเสจ) ให้แก่โปรแกรมเอง ดังนั้นรูปแบบการทำงานของโปรแกรมวินโดว์จึงเสมือนเป็นการโต้ตอบกับเมสเสจเท่านั้น ซึ่งลักษณะนี้เรียกว่า "message-driven" ซึ่งลักษณะนี้จะเป็นดังนี้ เมื่อระบบวินโดว์เรียกเข้าโพรซิเดอร์หลัก (main procedure) ของโปรแกรม โปรแกรมก็จะเริ่มต้นจัดการเกี่ยวกับตัวแปรและสร้างวินโดว์ หลังจากนั้นโปรแกรมก็จะเข้าวงจรรอบรอรับเมสเสจที่อยู่ใน queue จนกระทั่งเจอ 'WM_QUIT โปรแกรมจึงจะจบการทำงาน ถ้าเป็นเมสเสจอื่น ๆ ก็จะส่งกลับไปให้ระบบวินโดว์ ซึ่งวิธีทั้งหมดนี้เป็นไปตามขั้นตอนดังรูปที่ 4

โพรซิเดอร์ (Procedure)

ถ้าสังเกตโพรซิเดอร์หลัก จะเห็นว่าไม่มีส่วนไหนเลยที่ทำการโต้ตอบหรือนำเมสเสจไปประมวลผลบางอย่างเลย จะทำก็เพียงแค่รับเมสเสจและส่งกลับไปให้ระบบ ระบบจะเป็นส่วนจัดการส่งเมสเสจที่ได้รับมาจากโพรซิเดอร์หลักไปยัง "window procedure"



นั่นคือโพรซิเดอร์หลักเรียกโพรซิเดอร์วินโดว์ทางอ้อมโดยการส่งเมสเสจไปยังระบบก่อน และให้ระบบเป็นผู้เรียกโพรซิเดอร์วินโดว์ และโดยที่เมสเสจจะเป็นพารามิเตอร์ของโพรซิเดอร์วินโดว์ ลักษณะของโพรซิเดอร์วินโดว์จะเป็น switch และ case เช่นในรูปที่ 5

และเนื่องด้วยโพรซิเดอร์วินโดว์จะเลือกเอาเฉพาะเมสเสจอันหนึ่งเท่านั้น ดังนั้นจึงจำเป็นต้องมี default case เพื่อที่จะได้ส่งเมสเสจที่ไม่ต้องการกลับไปให้ระบบเป็นตัวจัดการต่อไป

นอกจากระบบจะต้องจัดการเกี่ยวกับเมสเสจแล้ว ยังต้องทำการจัดสรรหน่วยความจำให้แก่โปรแกรมทั้งหมดด้วย ทั้งส่วนของโค้ดและข้อมูลของโปรแกรม ระบบวินโดว์ได้กำหนดหมายเลขให้แก่โปรแกรมซึ่งเรียกว่า instance ทำให้การจัดสรรหน่วยความจำมีประสิทธิภาพ เมื่อไหร่ก็ตามที่มีการ

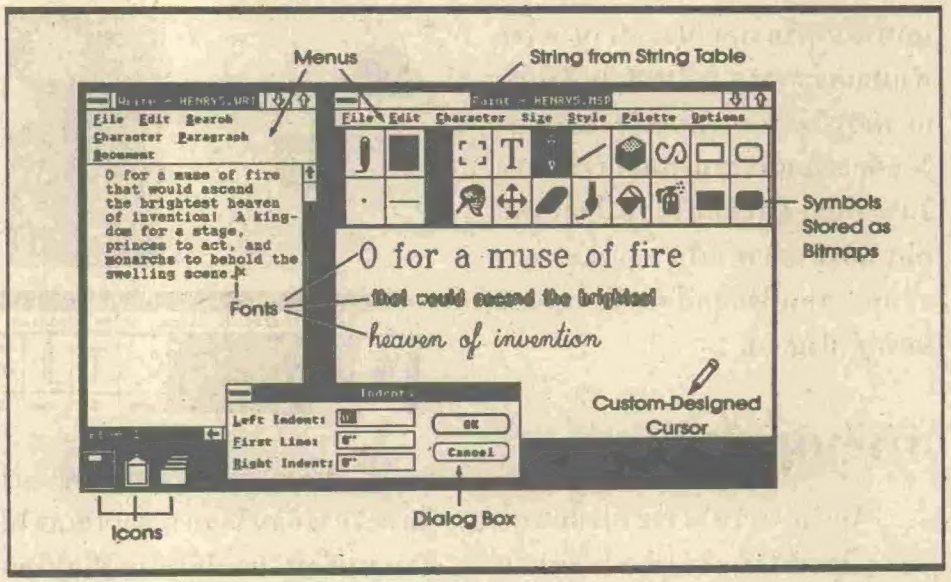
```
initialization

while the next message is not WM_QUIT
{
  forward the next message to the window it affects
}

end the program
```

รูปที่ 4

```
switch (type of message)
{
  case WM_SETFOCUS:
    display blinking caret at insertion point
    break;

  case WM_LBUTTONDOWN:
    move caret to new mouse position
    break;

  case WM_CHAR:
    insert new character at the caret
    break;

  case WM_PAINT:
    write the current page on the screen
    break;

  default:
    call default window procedure
    break;
}
```

รูปที่ 5

โหลดโปรแกรมซ้ำกับโปรแกรมที่ทำงานอยู่ในหน่วยความจำแล้วหมายเลขอินสแตนซ์จะเป็นตัวตรวจสอบเพื่อที่จะไม่ต้องโหลดโปรแกรมใหม่นั้น แต่จะให้ใช้โค้ดของโปรแกรมร่วมกันและส่วนของดาต้าเท่านั้นที่จะสร้างขึ้นมาใหม่ (share code multiple data) ระบบวินโดว์ยังทำการย้ายทั้งส่วนโค้ดและข้อมูลของโปรแกรมไปมาได้ เพื่อที่จะได้จัดการหน่วยความจำอย่างมีประสิทธิภาพ ดังนั้นการที่โปรแกรมจะอ่านหรือเขียนข้อมูลเข้าไปในหน่วยความจำจะต้องอ้างโดยใช้หมายเลข "handle" นอกจากนี้ระบบวินโดว์กันหน่วยความจำบางส่วนเพื่อเก็บข้อมูลที่เรียกว่า "resource" ซึ่งเป็นส่วนที่เก็บออปเจคต่าง ๆ ของระบบและของโปรแกรม เช่น เมนู ไอคอน ไดอะล็อก เป็นต้น ดังในรูปที่ 6



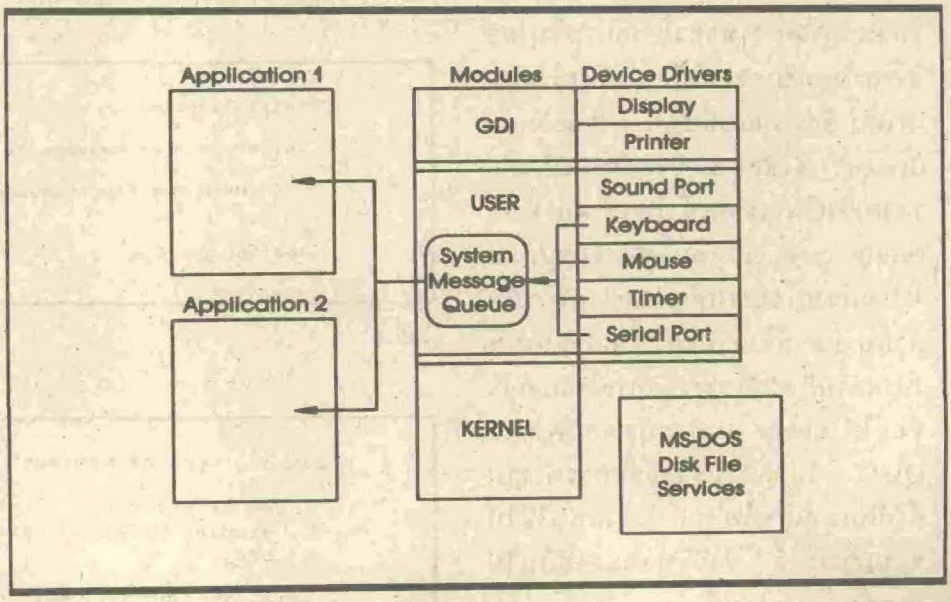
รูปที่ 6

แบบของโพรซิเดอร์ที่ติดต่อกับระบบวินโดว์จะถูกกำหนดเป็นแบบปาสคาล นั่นคือ คำนึงถึงการส่งผ่านพารามิเตอร์ การปรับสแต็กเมื่อกลับไปยังผู้เรียก ทั้งนี้การกำหนดแบบปาสคาลจะทำให้ประหยัดเนื้อที่ การกำหนดโพรซิเดอร์หลัก คือ WinMain และโพรซิเดอร์วินโดว์เป็นดังนี้

```
int PASCAL WinMain (.....)
long FAR PASCAL WndProc (.....)
```

ชื่อของโพรซิเดอร์หลักจะต้องเป็น WinMain ส่วนโพรซิเดอร์ระบบวินโดว์จะกำหนดเป็นอะไรก็ได้ ดังในตัวอย่างนี้คือ WndProc

ระบบวินโดว์ได้เอื้ออำนวยต่อการใช้โค้ดของโปรแกรมร่วมกัน ยังได้ให้ฟังก์ชันของระบบที่ให้โปรแกรมใช้ร่วมกันได้ด้วย ซึ่งลักษณะนี้เรียกว่า dynamic link" โดยทั่วไปในการพัฒนาโปรแกรม ฟังก์ชันใช้งานต่าง ๆ จะอยู่

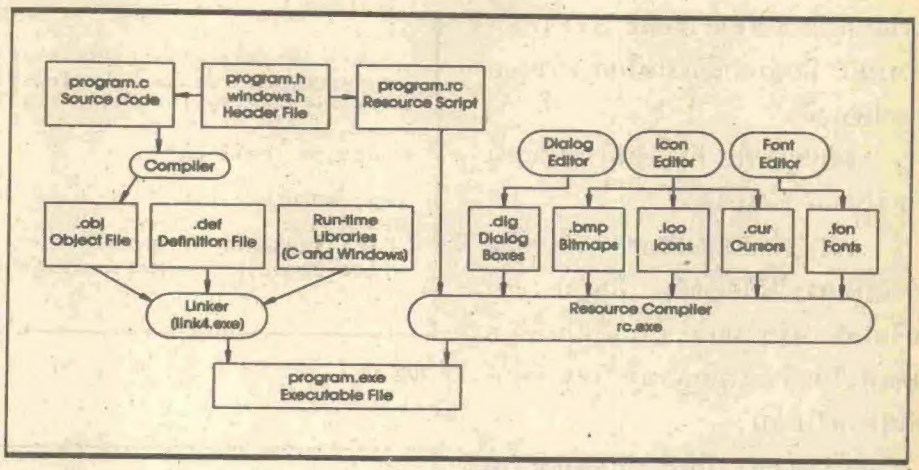


รูปที่ 7

ในไลบรารี่และจะถูกรวมเข้ากับโปรแกรมในขณะที่จะสร้างเป็น .EXE ไฟล์ซึ่งเรียกว่า "static link" แต่สำหรับไดนามิกลิงค์แล้วฟังก์ชั่นต่าง ๆ ไม่ได้ถูกรวมเข้ากับโปรแกรมในขณะที่สร้างไฟล์ .EXE ฟังก์ชันต่าง ๆ จะถูกเตรียมพร้อมอยู่ในหน่วยความจำโดยระบบเมื่อโปรแกรมถูกโหลดลงหน่วยความจำ ระบบจะเชื่อมโยงฟังก์ชันต่าง ๆ เข้ากับโปรแกรมเอง ซึ่งวิธีทำให้ไฟล์ .EXE ขนาดเล็กลง และยังทำให้ฟังก์ชันไม่ซ้ำซ้อนกัน

โครงสร้างโดยรวมของโปรแกรมและระบบเป็นไปดังรูปที่ 7 ส่วน GDI (Graphic Device Interface) เป็นส่วนของฟังก์ชันที่จัดการทางการแสดงผล

ทางจอภาพและเครื่องพิมพ์ ส่วนของ Kernel เป็นส่วนของฟังก์ชันที่จัดการหน่วยความจำ การโหลดของโปรแกรม การทำงานของโปรแกรมต่าง ๆ ส่วนของ User เป็นส่วนที่รับเมสเสจจากดีไวซ์ไดรเวอร์ เอาเข้า queue ของระบบและทำการส่งเมสเสจไป queue ของโปรแกรมต่าง ๆ ส่วนของ Device Driver ทำหน้าที่เป็นตัวกลางระหว่างฟังก์ชันต่าง ๆ ของระบบกับอุปกรณ์



รูปที่ 8

การเขียนโปรแกรมวินโดว์

หลังจากที่คุ้นเคยรูปร่างและโครงสร้างโดยรวมของระบบวินโดว์แล้ว เราก็จะมาศึกษาการเขียนโปรแกรมวินโดว์กัน ดังที่กล่าวไปแล้วว่าโปรแกรมวินโดว์เป็น "message-driven" ดังนั้นโครงสร้างของโปรแกรมวินโดว์จึงแตกต่างจากโปรแกรมโดยทั่วไป ขั้นตอนการสร้างโปรแกรมวินโดว์และไฟล์ที่จำเป็นต่าง ๆ เป็นไปดังรูปที่ 8

ไฟล์ในวินโดว์

header file เป็นไฟล์เฮดเดอร์เหมือนในภาษาซีโดยทั่วไป โดยที่จะกำหนดรูปแบบรีซอร์สของโปรแกรม แมคโคร และค่าคงที่ต่าง ๆ เช่น หมายเลขของโปรแกรม หมายเลขต่าง ๆ ในเมนู เป็นต้น ไฟล์นี้มีนามสกุล .H

resource script file เป็นไฟล์ที่อธิบายถึงออปเจคสำหรับโปรแกรมออปเจคต่าง ๆ อาทิเช่น ไดอะล็อกบ็อก, เมสเสจบ็อก, ไอคอนหรือเมนู เป็นต้น ไฟล์นี้มีสกุล .RC

definition file เป็นไฟล์ที่บอกถึงรายละเอียดการใช้หน่วยความจำ สแต็กและฮีพของโปรแกรม กำหนดลักษณะของโค้ดและข้อมูลของโปรแกรม นอกจากนี้ยังเป็นส่วนกำหนดโพรซีเดอร์วินโดว์ต่าง ๆ ที่ระบบวินโดว์จะเรียกได้โดยตรง ไฟล์นี้มีนามสกุล .DEF

source file เป็นโปรแกรมวินโดว์มีนามสกุล .C

```
/*****************************************************/
/*  HELLO.H: header file for HELLO.RC and HELLO.C */
/*****************************************************/

#define IDS_APPNAME 10
#define IDS_TITLE   20
```

รูปที่ 9

make file เป็นสคริปสำหรับโปรแกรม make เพื่อที่จะทำการคอมพายล์และลิงค์

ต่อไปจะเป็นตัวอย่างโปรแกรมซึ่งจะแสดงคำว่า "Hello World" บนจอภาพวินโดว์โดยจะแสดงรายละเอียดของโปรแกรมพร้อมคำอธิบาย โดยเริ่มจากไฟล์ .H ไฟล์ .RC ไฟล์ .DEF และไฟล์ .C ตามลำดับ

ไฟล์ hello.h กำหนดชื่อ IDS_APPNAME มีค่า 10 และ IDS_TITLE มีค่า 20 เราสามารถกำหนดค่าหมายเลขเหล่านี้ได้โดยอิสระ ชื่อเหล่านี้จะถูกอ้างอิงในไฟล์ hello.rc และ hello.c ดังรูปที่ 9

ไฟล์ hello.rc กำหนดข้อความตามชื่อหรือหมายเลขที่กำหนดในไฟล์

hello.h โดยใช้คีย์เวิร์ดคือ STRINGTABLE มีกลุ่มคำคีย์เวิร์ดในการกำหนดออปเจค คือ

MENU คือ คีย์เวิร์ดกำหนดเมนูของโปรแกรมวินโดว์

ACCELERATORS เป็นส่วนกำหนดการใช้คีย์บอร์ด คือปุ่ม ctrl หรือ alt หรือ shift ร่วมกับปุ่มอื่น ๆ โดยทั่วไปแล้วจะเป็นส่วนที่กำหนดเพิ่มเติมสำหรับเมนู

DIALOG เป็นส่วนอธิบายขนาดและข้อความที่จะแสดงในไดอะล็อกบ็อก

ICON เป็นส่วนการกำหนดรูปสัญลักษณ์เมื่อวินโดว์ถูกลดขนาด

CURSOR, FONT, BITMAP เป็นส่วนกำหนดรูปของเคอร์เซอร์ ฟอนต์ตัวอักษร และรูปบิตแม็พ ดังรูปที่ 10

ไฟล์ hello.def เป็นไฟล์ที่กำหนดสิ่งที่จำเป็นสำหรับโปรแกรม โดยมีคีย์เวิร์ดหลายคำ คือ

NAME กำหนดชื่อของโปรแกรม คือ hello

DESCRIPTION เป็นส่วนที่กำหนดข้อความซึ่งจะอยู่ในส่วนเฮดเดอร์ของโปรแกรมหรือไฟล์ hello.exe ซึ่งจุดนี้เป็นส่วนสำหรับการเก็บเวอร์ชันของโปรแกรมได้

STUB เป็นส่วนที่กำหนดชื่อของโปรแกรมคือ winstub.exe ซึ่งจะถูกรวมกับโปรแกรม hello.exe โปรแกรม winstub.exe จะแสดงข้อความ

This program requires Microsoft Windows

เมื่อโปรแกรม hello.exe ถูกเรียกภายนอกระบบวินโดว์

CODE และ DATA เป็นส่วนกำหนดพฤติกรรมของโปรแกรมในหน่วยความจำ โดยที่ส่วนโค้ดของโปรแกรมสามารถที่จะถูกย้ายไปมาได้

```
/*****************************************************/
/* HELLO.RC: resource script file for HELLO program */
/*****************************************************/

#include "hello.h"

STRINGTABLE
BEGIN
  IDS_APPNAME,    "Hello"
  IDS_TITLE,      "Hello World Application"
END
```

รูปที่ 10

```
;*************************************************
;*  HELLO.DEF: definition file for HELLO program  *
;*************************************************

NAME      Hello

DESCRIPTION 'first program for learning Microsoft Windows'

STUB      'WINSTUB.EXE'

CODE      MOVEABLE
DATA      MOVEABLE MULTIPLE

HEAPSIZE  1024       ;room for local variables
STACKSIZE 4096       ;room needed for function calls
                     ; 4k bytes as minimum statcksize

EXPORTS              ;you must export all procedures called by window
  HelloWinProc   @1 ;(ordinal number)
```

รูปที่ 11

(MOVEABLE) และดาต้าสามารถที่จะถูกย้ายและมีได้หลายชุด (MULTIPLE) นั่นคือ ในแต่ละอินสแตนซ์ของโปรแกรมส่วนโค้ดจะใช้ร่วมกัน แต่จะมีส่วนดาต้าที่มีหลายชุด

HEAPSIZE และ STACKSIZE กำหนดขนาดของฮีพและสแต็กของโปรแกรม โดยปกติสแต็กจะต้องมีขนาดอย่างน้อย 4 กิโลไบต์

EXPORTS เป็นส่วนที่บอกลิงเกอร์ว่ามีโพรซิเดอร์ระบบวินโดว์ ชื่อ HelloWinProc และตัวอักษร @1 หมายถึงว่า โพรซิเดอร์วินโดว์นี้เป็นเอกซ์พอร์ตโพรซิเดอร์หมายเลข 1 ดังรูปที่ 11

สำหรับไฟล์ hello.c ประกอบด้วยหลาย ๆ โพรซิเดอร์ คือ WinMain, HelloInitApp, HelloFirstInit, HelloWinProc และ HelloPaint ซึ่งได้บอกกล่าวไว้ตอนต้นของไฟล์

WinMain เป็นจุดเริ่มต้นของโปรแกรม โดยมีแบบเป็น int PASCAL โพรซิเดอร์นี้มีพารามิเตอร์ที่ส่งผ่านดังนี้

HANDLE hInstance, hPrevInstance เป็นค่าตัวเลขอินสแตนซ์ของโปรแกรมและค่าอินสแตนซ์ของโปรแกรมก่อน

LPSTR lpszCmdLine เป็นพอยน์เตอร์ที่ชี้ข้อความคอมมานด์ไลน์

int CmdShow เป็นพารามิเตอร์

ที่ใช้สำหรับฟังก์ชัน ShowWindow

ตอนแรก WinMain จะเรียก HelloInitApp จากนั้นจะทำการวนรอบรอบโดยเรียกฟังก์ชัน GetMessage ซึ่งฟังก์ชันนี้จะดึงเมสเสจมาจาก queue ของโปรแกรมมาไว้ในตัวแปร &msg GetMessage จะส่งค่า TRUE กลับ และส่งค่า FALSE ซึ่งจะทำให้หลุดจากการทำงานรอบเมื่อเมสเสจเป็น WM_QUIT ฟังก์ชัน TranslateMessage ทำการแปลงเมสเสจที่เกี่ยวกับคีย์บอร์ด และ DispatchMessage จะทำการส่งเมสเสจกลับไปยังระบบ และระบบจะส่งไปยัง HellowWinProc อีกทอดหนึ่ง

โพรซิเดอร์ HelloInitApp ทำการสร้างวินโดว์ของโปรแกรม โดยอันดับแรกจะทำการเรียก HelloFirstInit ถ้าหากว่าโปรแกรมเป็นอินสแตนซ์แรก โดยที่ hPrevInstance จะเป็นศูนย์ แต่ถ้าไม่ใช่ก็จะอ่านข้อความซึ่งเป็นรีซอร์สโดยใช้ฟังก์ชัน GetOmstamceData ซึ่งข้อความจะเก็บอยู่ที่ szAppName และ szTitle หลังจากนั้นจะทำการสร้างวินโดว์โดยเรียกฟังก์ชัน Create Window ซึ่งจะส่งหมายเลขประจำวินโดว์กลับไว้ใน hWnd ส่วนฟังก์ชัน ShowWindow จะทำการแสดงวินโดว์ไปบนจอภาพ สำหรับฟังก์ชัน UpdateWindow จะเป็นการส่งเมสเสจ WM_PAINT ลงไปใน quene ของโปรแกรม เพื่อที่จะให้ HelloWinProc รับเมสเสจนี้ไปแสดงผลในไคลต์แอเรีย

โพรซิเดอร์ HelloFirstInit จะทำการโหลดข้อความจากรีซอร์ส โดยใช้ฟังก์ชัน LoadString และทำการเรียกฟังก์ชัน Register Class

ฟังก์ชัน LoadCursor กำหนด

รูปที่ 12

```
/*********************************************************/
/* HELLO.C: source program displays "HELLO WORLD" */
/*********************************************************/

#include <windows.h>
#include "hello.h"

/*********************************************************/
/*              declaration functions                    */
/*********************************************************/

int PASCAL      WinMain(HANDLE, HANDLE, LPSTR, int);
BOOL            HelloInitApp(HANDLE, HANDLE, int);
BOOL            HelloFirstInit(HANDLE);
long FAR PASCAL HelloWinProc(HWND, WORD, WORD, LONG);
void            HelloPaint(HWND);

/*********************************************************/
/*                  GLOBAL VARIABLES                     */
/*********************************************************/

char szAppName[10];  /* buffer to hold application name   */
char szTitle[25];    /* buffer to hold caption for window */

/*********************************************************/
/*                  MAIN PROCEDURE                       */
/*********************************************************/

int PASCAL WinMain(hInstance, hPrevInstance, lpszCmdLine, CmdShow)
HANDLE hInstance, hPrevInstance;
LPSTR lpszCmdLine;          /* points to command line   */
int CmdShow;                /* iconic or tiled at start */
{
  MSG msg;                  /* temp buffer for message */

  /* initialization routine */
  if (!HelloInitApp(hInstance, hPrevInstance, CmdShow))
    return (FALSE);

  /* main loop, terminated by a quit message */
  while (GetMessage(&msg, NULL, 0, 0)) {
    /* have Windows translate the message */
    TranslateMessage(&msg);
    /* have Windows pass the message on to the window procedure */
    DispatchMessage(&msg);
  }

  /* you must return the word parameter        */
  /* from the WM_QUIT message as as exit code */
  return (msg.wParam);
}

/*********************************************************/
/*          INITIALIZATION PROCEDURE                     */
/*********************************************************/

BOOL HelloInitAPp(hInstance, hPrevInstance, CmdShow)
HANDLE hInstance, hPrevInstance;
int CmdShow;
{
  HWND hWnd; /* a handle for the widnow belonging to this instance */

  /* if this application is not already running in another window... */
  if (!hPrevInstance) {
    /* ...then do first-instance initialization...*/
    if (!HelloFirstInit(hInstance))
```

ลักษณะของเคอร์เซอร์ให้กับวินโดว์ซึ่งก็จะเป็นรูปลูกศร (IDC_ARROW) และฟังก์ชัน (WHITE_BRLSH)

โพรซิเดอร์ HelloWinProc จะรับเมสเสจจากระบบโดยที่กำหนดแบบเป็น long FAR PASCAL และจะมีพารามิเตอร์ 4 ค่าส่งผ่านเข้ามาคือ

HWND hWnd เป็นหมายเลขกำหนดวินโดว์

WORD msg เป็นตัวเลขกำหนดเมสเสจ เช่น ค่า 0x000F คือ WM_PAINT

WORD wParam เก็บค่า 2 ไบต์ และ LONG lParam เก็บค่า 4 ไบต์ ซึ่งพารามิเตอร์สองตัวสุดท้ายนี้จะขึ้นอยู่กับชนิดของเมสเสจ เช่น wParam เป็นค่าแอสกีของคีย์บอร์ดเมื่อเมสเสจเป็น WM_CHAR หรือ BUTTONDOWN ค่า wParam จะเป็นปุ่มที่คลิกและ lParam จะเป็นตำแหน่งของเมาส์

HelloWinProc จะทำการแปลความหมายตามชนิดของเมสเสจ คือ ถ้าเป็น

WM_PAINT จะทำการเรียก Hellopaint ซึ่งจะแสดงข้อความ

WM_DESTROY จะส่งเมสเสจ WM_QUIT โดยฟังก์ชัน PostQuit Message

ส่วนเมสเสจอื่น ๆ จะถูกส่งกลับไปยังระบบโดยฟังก์ชัน DefWinProc

โพรซิเดอร์ HelloPaint จะทำการแสดงข้อความโดยที่เรียกฟังก์ชัน BeginPaint เพื่อที่จะได้ display context เก็บไว้ในตัวแปร hDC ซึ่งค่า display context นี้เป็นหมายเลขที่กำหนดให้กับฟังก์ชันต่าง ๆ ของ GDI จากนั้นก็จะนำข้อความขึ้นจอภาพโดยที่ตำแหน่ง

รูปที่ 12 ต่อ

```
        return (FALSE);
    } else {
      /* ...otherwise, copy data from the previous instance. */
      GetInstanceData(hPrevInstance, szAppName, 10);
      GetInstanceData(hPrevInstance, szTitle, 25);
    }

    /* Make a class "Hello" window exist for this instance */
    hWnd = CreateWindow(
      szAppName,            /* Window class name          */
      szTitle,              /* string for window caption */
      WS_OVERLAPPEDWINDOW, /* window style               */
      /* let Windows set positon and size of window     */
      CW_USEDEFAULT,        /* horz postion of window     */
      CW_USEDEFAULT,        /* vert postion of window     */
      CW_USEDEFAULT,        /* width of window            */
      CW_USEDEFAULT,        /* height of window           */
      NULL,                 /* no parent window           */
      NULL,                 /* no menu for this window    */
      hInstance,            /* who created this window    */
      NULL);                /* no parameter to pass on    */

    /* Make the window visible */
    ShowWindow(hWnd, CmdShow);

    /* Send yourself a WM_PAINT message and force */
    /* the paint procedure to run immediately     */
    UpdateWindow(hWnd);

    return (TRUE);  /* initilization succeeded */
}

/*************************************************/
/*          FIRST INSTANCE INITIALIZATION        */
/*************************************************/

BOOL HelloFirstInit(hInstance)
HANDLE hInstance;
{
    /* use the WNDCLASS structure for defining window class */
    WNDCLASS wc;

    /* load strings from resource */
    LoadString(hInstance, IDS_APPNAME, szAppName, 10);
    LoadString(hInstance, IDS_TITLE, szTitle, 25);

    /* fill structure with information about this window class */
    wc.lpszClassName = szAppName;
    wc.hInstance = hInstance;
    wc.lpfnWndProc = HelloWinProc;
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hIcon = NULL;
    wc.lpszMenuName = NULL;
    wc.hbrBackground = GetStockObject(WHITE_BRUSH);
    wc.style = CS_HREDRAW|CS_VREDRAW;
    wc.cbClsExtra = 0;
    wc.cbWndExtra = 0;

    /* tell Windows about your new window class */
    if (!RegisterClass(&wc))
      return (FALSE); /* initialization failed */

    return (TRUE);  /* initialization succeeded */
}

/*************************************************/
```

รูปที่ 12 ต่อ

```
/*                WINDOW PROCEDURE                    */
/****************************************************/

long FAR PASCAL HelloWinProc(hWnd, msg, wParam, lParam)
HWND hWnd;
WORD msg;
WORD wParam;
LONG lParam;
{
  switch (msg) {
    case WM_PAINT:
      /* The window needs to be redrawn */
      HelloPaint(hWnd);
      break;
   case WM_DESTROY:
     /* Windows has removed this window from screen */
     PostQuitMessage(0); /* send a quit message to end WinMain */
     break;
   default:
     /* The messages you don't handle yourself come here */
     /* You must put this call in your window procedure  */
     return (DefWindowProc(hWnd, msg, wParam, lParam));
  }

  /* A window procedure should return 0 to indicate success */
  return (0L);
}


/****************************************************/
/*              HELLOPAINT PROCEDURE                */
/****************************************************/

void HelloPaint(hWnd)
HWND hWnd;
{
  PAINTSTRUCT ps;
  HDC hDC;

  /* establish a display context */
  hDC = BeginPaint(hWnd, &ps);

  /* Show string on client area at (100,100) */
  TextOut(hDC,100,100,"Hello World!",12);

  /* discard any pending WM_PAINT messages because the screen is uptodate */
  ValidateRect(hWnd, NULL);

  /* release memory allocation for the paint procedure */
  EndPaint(hWnd, &ps);
}
```

(100,100) โดยเรียกฟังก์ชัน TextOut เมื่อแสดงข้อความเสร็จแล้ว จึงเรียก EndPaint เพื่อคืน display context ให้กับระบบ ดังรูปที่ 12

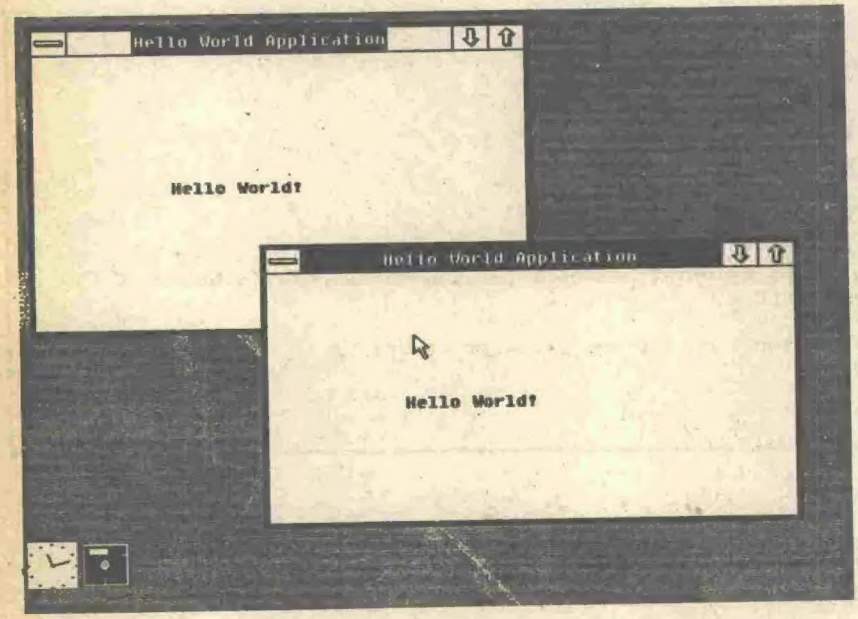
การคอมพายล์และลิงค์นั้น สามารถทำได้โดยป้อนคำสั่งตรง ๆ หรือจะสร้างสคริปต์ไฟล์ สำหรับโปรแกรม make ดังในรูปที่ 13 ซึ่งโดยความสะดวกแล้วขอแนะนำให้ใช้วิธีหลังจะสะดวกกว่า เพราะว่ามีออปชันในการคอมพายล์มาก

ผลลัพธ์ของโปรแกรม Hello.exe เป็นดังรูปที่ 14

บทความนี้ได้กล่าวถึงโครงสร้างระบบวินโดว์ของไมโครซอฟต์ และการพัฒนาโปรแกรมที่ทำงานภายใต้วินโดว์ของไมโครซอฟต์ โดยใช้ชุดพัฒนาซอฟต์แวร์ของไมโครซอฟต์คือ SDK

```
#*********************************************************
#* HELLO : make file for maintenace the HELLO program    *
#*********************************************************

#
#  Standard command line definitions
#

cp=cl -d -c -AS -Gsw -Os -Zpe

#
#  Standard inference rules
#
.c.obj:
    $(cp) $*.c

#
#  The C File List
#

hello.obj: hello.c hello.h

hello.res: hello.rc hello.h
    rc -r hello.rc

hello.exe: hello.obj hello.res hello.def
    link4 hello,/align:16,/map,slibw slibcew,hello.def
    mapsym hello
    rc hello.res
```

รูปที่ 13



รูปที่ 14

(Software Development Toolkit)

จะเห็นได้ว่าการเขียนโปรแกรมภายใต้ระบบวินโดว์ของไมโครซอฟต์ไม่ได้เป็นเรื่องยุ่งยากอย่างที่คิด เพราะมีเครื่องมือช่วยต่าง ๆ มากมาย ทำให้เราไม่ต้องมาพัฒนาส่วนยูสเซอร์อินเตอร์เฟสกันใหม่ โปรแกรมที่ได้ก็มีคุณภาพสูง การใช้งานก็ง่ายโดยเราไม่ต้องลงแรงมาก ข้อดีเหล่านี้ทำให้การพัฒนาโปรแกรมในระบบวินโดว์จะมีอิทธิพลต่อวงการอย่างใหญ่หลวงในอนาคตอันใกล้นี้